Государственное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н.Э. Баумана»

Отчет
По лабораторной работе №2
По курсу «Конструирование компиляторов»
На тему
«Преобразования грамматик»

Студент: Мурашов И.Д.
Группа: ИУ7-23М
Вариант: 4
Преподаватель: Ступников А.А.

Москва, 2020

Оглавление

1 Цель и задачи работы

Цель работы: приобретение практических навыков реализации наиболее важных (но не всех) видов преобразований грамматик, чтобы удовлетворить требованиям алгоритмов синтаксического разбора.
Задачи работы:

1. Принять к сведению соглашения об обозначениях, принятые в литературе по теории формальных языков и грамматик и кратко описанные в приложении.

2. Познакомиться с основными понятиями и определениями теории формальных языков и грамматик.

3. Детально разобраться в алгоритме устранения левой рекурсии.

4. Разработать, тестировать и отладить программу устранения левой рекурсии.

5. Разработать, тестировать и отладить программу преобразования грамматики в соответствии с предложенным вариантом.

2 Текст программы

```
        using Lab2_Grammar.Elements;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;

namespace Lab2_Grammar
{
public class Program
{
public static void Main(string[] args)
{
//      G0
Dictionary<NonTerminal, List<Generation>> productions = XmlWorker.Gra
out List<NonTerminal> nonTerminals);
SaveReadableFormat("grammar_in.txt", terminals, nonTerminals, product

productions = GrammarWorker.LRElimination(terminals, nonTerminals, pr
SaveReadableFormat("grammar_out_noLR.txt", terminals, nonTerminals, p
XmlWorker.GrammarWriter("Grammar_noLR.xml", terminals, nonTerminals,

//      G1
Dictionary<NonTerminal, List<Generation>> productions2 = XmlWorker.Gr
out List<NonTerminal> nonTerminals2);
SaveReadableFormat("grammar_in2.txt", terminals2, nonTerminals2, proc

productions2 = GrammarWorker.LRElimination(terminals2, nonTerminals2,
SaveReadableFormat("grammar_out_noLR2.txt", terminals2, nonTerminals2
XmlWorker.GrammarWriter("Grammar_noLR2.xml", terminals2, nonTerminals

//          G0

Dictionary<NonTerminal, List<Generation>> productions3 = XmlWorker.Gr
out List<NonTerminal> nonTerminals3);
SaveReadableFormat("grammar_in3.txt", terminals3, nonTerminals3, proc

productions3 = GrammarWorker.chainRuleElimination(terminals3, nonTerr
SaveReadableFormat("grammar_out_noCR3.txt", terminals3, nonTerminals3
XmlWorker.GrammarWriter("Grammar_noCR3.xml", terminals3, nonTerminals
productions3 = GrammarWorker.chainRuleElimination(terminals3, nonTerr
SaveReadableFormat("grammar_out_noLR3.txt", terminals3, nonTerminals3
XmlWorker.GrammarWriter("Grammar_noLR3.xml", terminals3, nonTerminals


}

private static void SaveReadableFormat(string filename, List<Terminal
Dictionary<NonTerminal, List<Generation>> productions)
```

```
{
StringBuilder terms = new StringBuilder("T: ");
foreach (var terminal in terminals)
terms.Append($"{terminal.Spell}, ");

StringBuilder nonterms = new StringBuilder("NT: ");
string start = "";
foreach (var nonTerminal in nonTerminals)
{
nonterms.Append($"{nonTerminal.Name}, ");
if (nonTerminal.IsStart)
start = nonTerminal.Name;
}

List<StringBuilder> sProductions = new List<StringBuilder>();
foreach (var produciton in productions)
{
StringBuilder sProduction = new StringBuilder($"{produciton.Key.Name}
foreach (var generation in produciton.Value)
{
sProduction.Append(generation.GenString);
sProduction.Append(" | ");
}
sProductions.Add(sProduction);
}

using (StreamWriter sw = new StreamWriter(filename, false))
{
sw.WriteLine(terms.ToString());
sw.WriteLine(nonterms.ToString());
foreach (var sProduction in sProductions)
sw.WriteLine(sProduction.ToString());
sw.WriteLine(start);
}
}
}
}

using Lab2_Grammar.Elements;
using Lab2_Grammar.Helpers;
using System;
using System.Collections.Generic;
using System.Linq;

namespace Lab2_Grammar
{
public static class GrammarWorker
{

public static Dictionary<NonTerminal, List<Generation>> LREliminatio
```

```
List<NonTerminal> nonTerminals, Dictionary<NonTerminal, List<Generati
{
// C
var productionNonTerms = productions.Keys.ToList();
int ntCount = productionNonTerms.Count;

for (int i = 0; i < ntCount; i++)
{
//    Ai
List<Generation> currGens = new List<Generation>(productions[producti

for (int j = 0; j < i; j++)
{
int gensCount = currGens.Count;
for (int k = 0; k < gensCount; k++)
{
//    k-  Ai
Generation gen = new Generation(currGens[k]);

//      Ai -> Aja
if (gen.Gen.First().Name == productionNonTerms[j].Name)
{
//    Aj
List<Generation> jGens = new List<Generation>(productions[productionN

//    k- ,        Aj
currGens.RemoveAt(k);
//     (Aj)  k- ,       Aj
gen.Gen.RemoveAt(0);  //  a

//      Aj
foreach (var jGen in jGens)
{
//     Aj
Generation newJGen = new Generation(jGen);
//     k-  Ai     ( )
newJGen.Gen.AddRange(gen.Gen);
//         Ai
currGens.Add(newJGen);
}
}
}
}

//         Ai
productions.Remove(productionNonTerms[i]);

//
if (currGens.Any(g => g.Gen.First().Name == productionNonTerms[i].Nan
{
```

```
//          ' (Ai  Ai')
nonTerminals.Add(new NonTerminal() { Name = productionNonTerms[i].Nan
//       Ai'
List<Generation> newNTGens = new List<Generation>();
//       Ai
List<Generation> oldNTGens = new List<Generation>();

//    Ai  (Ai -> Aia | b)
foreach (var gen in currGens)
{
//      =>    Ai' -> aAi'
if (gen.Gen.First().Name == productionNonTerms[i].Name)
{
//   ""  ( Ai)
gen.Gen.RemoveAt(0);

//
Generation newGen = new Generation();
//            (a)
newGen.Gen.AddRange(gen.Gen);
//         (Ai')
newGen.Gen.Add(nonTerminals.Last());
newNTGens.Add(newGen);
}
else  //      =>    Ai -> bAi'
{
//
Generation newGen = new Generation();
//            (b)
newGen.Gen.AddRange(gen.Gen);
//         (Ai')
newGen.Gen.Add(nonTerminals.Last());
oldNTGens.Add(newGen);
}
}
//         Ai'  Ai' -> eps
newNTGens.Add(new Generation() { Gen = new List<Symbol>() { terminals

//          Ai
productions.Add(productionNonTerms[i], oldNTGens);
//          Ai'
productions.Add(nonTerminals.Last(), newNTGens);
}
else  //      -
{
//           Ai
productions.Add(productionNonTerms[i], currGens);
}
}
```

```
return productions;
}


public static Dictionary<NonTerminal, List<Generation>> chainRuleElim
List<NonTerminal> nonTerminals, Dictionary<NonTerminal, List<Generati
{
//  Na, Nb, Nc ... , a,b,c -
var Nlit = new Dictionary<NonTerminal, HashSet<NonTerminal>>();

//
for (int i = 0; i < nonTerminals.Count; i++)
{
// Na = {B|A -> B}
var N = new List<HashSet<NonTerminal>>();
//   N0
N.Add(new HashSet<NonTerminal>() { nonTerminals[i]} );
int j = 1;

//   C,  B->C
foreach (var production in productions)
{
//  B  N-1
if (N[j - 1].Contains(production.Key))
{
foreach (var value in production.Value)
{
//    C,  B->C
if (value.Gen.Count == 1 && nonTerminals.Contains(value.Gen[0]))
{
//   c Ni C
var tmp = new HashSet<NonTerminal>() { (NonTerminal)value.Gen[0] };
var tmp2 = N[j - 1];
//  Nj  Nj-1
tmp.UnionWith(tmp2);
//   Na
N.Add(tmp);

//   Nj   Nj-1 -
if (!N[j].SetEquals(N[j - 1]))
j++;
}

}
}

}
//    Na   Nj
Nlit.Add(nonTerminals[i], N[N.Count - 1]);
}
```

```
var newProductions = new Dictionary<NonTerminal, List<Generation>>();

//
foreach (var production in productions)
{

foreach (var value in production.Value)
{
//
if (!(value.Gen.Count == 1 && nonTerminals.Contains(value.Gen[0])))
{
//   Na
foreach (var Nl in Nlit)
{
//   Na
if (Nl.Value.Contains(production.Key))
{
if (newProductions.ContainsKey(Nl.Key))
{
newProductions[Nl.Key].Add(value);
}
else
{
newProductions.Add(Nl.Key, new List<Generation>() { value });
}
}
}
}
}
}

return newProductions;
}

}
}
using Lab2_Grammar.Elements;
using System.Collections.Generic;
using System.Linq;
using System.Xml.Linq;

namespace Lab2_Grammar
{
public static class XmlWorker
{
public static Dictionary<NonTerminal, List<Generation>> GrammarReader
out List<NonTerminal> nonTerminals)
{
var grammarXml = XDocument.Load(filename);
```

```
// Read terms and nonterms in temp vars
List<Terminal> terminalsT = grammarXml.Descendants("term").Select(i =
{
Name = (string) i.Attribute("name"),
Spell = (string) i.Attribute("spell")
}).ToList();
List<NonTerminal> nonTerminalsT = grammarXml.Descendants("nonterm").S
{
Name = (string) i.Attribute("name")
}).ToList();

// Save start symbol index
nonTerminalsT.Find(nt => nt.Name == (string) grammarXml.Root.Element(

Dictionary<NonTerminal, List<Generation>> productions = new Dictionar
// Read productions
foreach (var production in grammarXml.Descendants("production"))
{
NonTerminal nonTerm = nonTerminalsT.Find(t => t.Name == (string) proc
Generation generation = new Generation
{
Gen = production.Element("rhs").Elements().Select((XElement i) =>
{
if ((string) i.Attribute("type") == "term")
return (Symbol) terminalsT.Find(t => (string) i.Attribute("name") ==
else if ((string) i.Attribute("type") == "nonterm")
return (Symbol) nonTerminalsT.Find(nt => (string) i.Attribute("name")
else
return null;
}).ToList()
};

if (productions.TryGetValue(nonTerm, out List<Generation> gens))
{
gens.Add(generation);
productions.Remove(nonTerm);
productions.Add(nonTerm, gens);
}
else
{
productions.Add(nonTerm, new List<Generation>() { generation });
}
}

terminals = terminalsT;
nonTerminals = nonTerminalsT;
return productions;
}
```

```
public static void GrammarWriter(string filename, List<Terminal> term
Dictionary<NonTerminal, List<Generation>> productions)
{
XElement terminalsymbols = new XElement("terminalsymbols");
foreach (var terminal in terminals)
terminalsymbols.Add(new XElement("term", new XAttribute("name", termi

XElement nonterminalsymbols = new XElement("nonterminalsymbols");
XElement startsymbol = null;
foreach (var nonterminal in nonTerminals)
{
if (nonterminal.IsStart)
startsymbol = new XElement("startsymbol", new XAttribute("name", nont

nonterminalsymbols.Add(new XElement("term", new XAttribute("name", no
}

XElement xProductions = new XElement("productions");
foreach (var produciton in productions)
{
XElement lhs = new XElement("lhs", new XAttribute("name", produciton.

foreach (var generation in produciton.Value)
{
XElement rhs = new XElement("rhs");
foreach (var symbol in generation.Gen)
{
rhs.Add(new XElement("symbol", new XAttribute("type", symbol is Term
new XAttribute("name", symbol.Name)));
}

xProductions.Add(new XElement("production", lhs, rhs));
}
}

XElement grammar = new XElement("grammar", terminalsymbols, nontermin
new XAttribute("name", "G"));
XDocument xDoc = new XDocument(grammar);
xDoc.Save(filename);
}
}
}
```

3 Проверка правильности программы

3.1 Тест устранения левой рекурсии:

Входная грамматика:
Начальный символ: S **Терминалы**: a, +, *, (,), eps
Нетерминалы: E, T, F

Продукции:
E -> E+T | T
T -> T*F | F
F -> a | (E)
Выходная грамматика:
Терминалы: a, +, *, (,), eps
Нетерминалы: E, T, F, E', T'
Продукции:
E -> TE'
T -> FT'
F -> a | (E)
E' -> +TE' | eps
T' -> *FT' | eps

Входная грамматика:
Начальный символ: S **Терминалы**: a, b, c, d, eps
Нетерминалы: S, A
Продукции:
S -> Aa | b | a
A -> Ac | Sd | c
Выходная грамматика:
Терминалы: a, +, *, (,), eps
Нетерминалы: S, A, A'
Продукции:
S -> Aa | b | a
A -> cA' | bdA' | adA'
A' -> cA' | adA' | eps

3.2 Тест устранения цепных правил:

Входная грамматика:
Начальный символ: S **Терминалы**: a, +, *, (,)
Нетерминалы: E, T, F
Продукции:
E -> E+T | T
T -> T*F | F
F -> a | (E)
Выходная грамматика:
Терминалы: a, +, *, (,), eps
Нетерминалы: E, T, F
Продукции:
E -> E+T | T*F | a | (E)
T -> T*F | a | (E)
F -> a | (E)

4 Выводы

По результатам проведенной работы были приобретены практические навыки реализации практических навыков реализации наиболее важных (но не всех) видов преобразований грамматик, чтобы удовлетворить требованиям алгоритмов синтаксического разбора. Были приняты к сведению соглашения об обозначениях, принятые в литературе по теории формальных языков и грамматик и кратко описанные в приложении. Было проведено ознакомление с основными понятиями и определениями теории формальных языков и грамматик. Также была разработана, отестирована отлажана программа устранения левой рекурсии и удаления цепных правил.

5 Список литературы

1. АХО А.б УЛЬМАН Дж. Теория синтакстического анализа, перевода и компиляции: В 2-х томах. Т1.: Синтаксический анализ. - М.: Мир, 1978.
2. АХО А.В, ЛАМ М.С., СЕТИ Р., УЛЬМАН Дж.Д. Компиляторы: принципы, технологии и инструменты. – М.: Вильямс, 2008.